Laurea Specialistica in Informatica
a.a. 2005-2006

Interazione Uomo-Macchina II:

Interfacce Intelligenti

Fiorella de Rosis

Introduzione

Prima parte: *Formalizzazione e Ragionamento*

1.1. Ragionamento logico:
- Formalizzazione
- Risoluzione

1.2. Ragionamento incerto
- Reti Causali Probabilistiche
- Reti dinamiche
- Apprendimento di Reti

Seconda parte: *Modelli di Utente*

2.1. Modelli logici

2.2. Modelli con incertezza

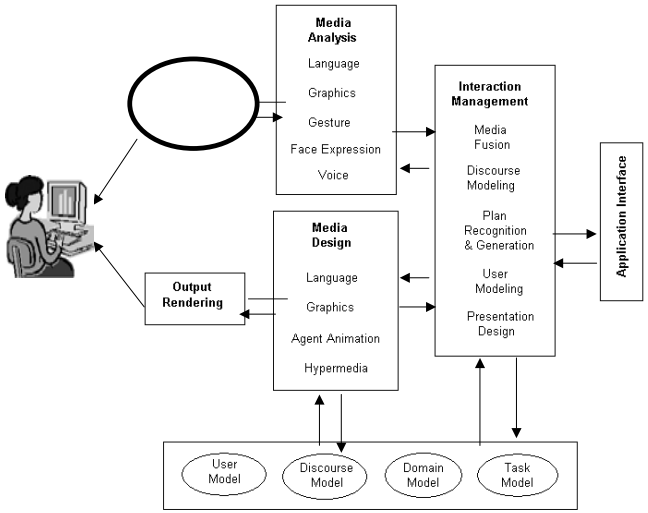
Terza parte: *Interazione in linguaggio naturale*

3.1. Generazione di messaggi
- Introduzione
- Teorie
- Metodi

3.2. Comprensione di messaggi

Quarta parte: *Simulazione di dialoghi*

Architettura di una interfaccia intelligente



Natural Language Understanding

(Comprensione e interpretazione del linguaggio naturale)

Nell'analisi del linguaggio si distingue, classicamente, fra:

- *Morfologia:* studio della **struttura delle parole**
- *Sintassi:* studio della **struttura delle frasi**
- *Semantica:* studio del **significato delle frasi**
- *Pragmatica:* studio degli **obiettivi comunicativi** delle frasi

Morfologia: Le parole possono essere categorizzate in 'Parts of Speech':

Parts of speech	Esempio
Nouns (noun)	dog, equation, concert
Pronouns (pro)	I, you, it, they, them
Possessive (pos)	my, your
Verb (verb)	is, had, went, loves
Adjective (adj)	red, large, lovely
Determiner (det)	the, a, some
Proper noun (prop)	Alice, John
Conjunction (conj)	and, but, since
Preposition (prep)	In, to, into
Auxiliary verb (aux)	be, have
Modal verb (modal)	will, can, must, should
Adverb (adv)	closely, quickly,
Wh-word (wh)	who, what, where,
Final punctuation (fpunct)	. , ? !

Esempio per l'inglese

Le 'parts of speech' si chiamano anche 'simboli lessicali'

Sintassi:
La struttura delle frasi in un determinato linguaggio può essere definita con una Grammatica

Le Grammatiche 'context-free' (CFG) permettono di definire la struttura delle frasi 'corrette' in un linguaggio naturale semplice.

Una CFG è composta da:
- un insieme di *simboli terminali*
- un insieme di *simboli non terminali*
- un non terminale designato come *'simbolo iniziale' (o 'seme')*
- un insieme di *regole di produzione* che hanno un unico non-terminale a sinistra ed una stringa di simboli terminali e/o non terminali a destra

Un esempio semplice di CFG

Simboli non terminali: le parts of speech che abbiamo visto prima, oltre a categorie più ampie: np, vp, s, s-maj e al *Seme:* s-maj

Regole:
1. s-maj → s fpunc
2. s → np vp
3. np → noun
4. np → det noun
5. vp → verb
6. vp → verb np
7. vp → verb np np
8. det → the
9. noun → salespeople
10. noun → dog
11. noun → biscuits
12. verb → ate
13. verb → sold
14. fpunc → .

Questa grammatica permette, ad Esempio, di generare la frase:

"Salespeople sold the dog biscuits"

Ma anche (estendendo il lessico):

"John sent Mary an email"

e simili

Usi delle Grammatiche

Le grammatiche possono essere utilizzate:

In modo generativo, per generare tutte e sole le stringhe che appartengono ad un linguaggio

In modo riconoscitivo, per riconoscere e classificare le frasi appartenenti a un linguaggio.

Uso generativo: Derivazione di una frase

a partire dal seme, si applicano le regole della grammatica una alla volta e in una sequenza opportuna, fino a ottenere la frase.

Un esempio di derivazione

"Salespeople sold the dog biscuits"

1. s-maj → s fpunc
2. s → np vp
3. np → noun
4. np → det noun
5. vp → verb
6. vp → verb np
7. vp → verb np np
8. det → the
9. noun → Salespeople
10. noun → dog
11. noun → biscuits
12. verb → ate
13. verb → sold
14. fpunc → .

Applico le regole una alla volta, a partire dal seme, fino ad ottenere la stringa:

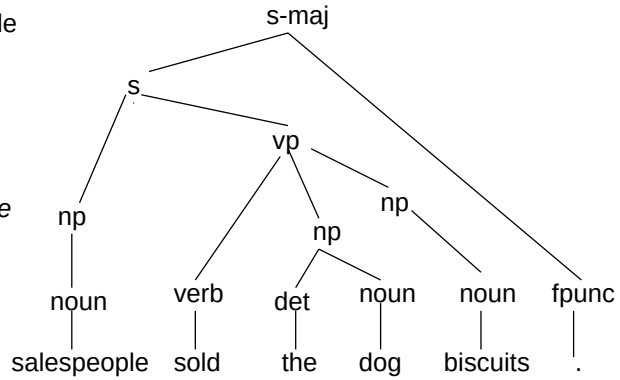
s-maj (1)⇒ s fpunc (14)⇒ s . (2)⇒
np vp . (3)⇒
noun vp fpunc (9)⇒ Salespeople vp . (7)⇒
Salespeople verb np np . (13)⇒
Salespeople sold np np . (4)⇒
Salespeople sold det noun np . (8) ⇒
Salespeople sold the noun np . (10) ⇒
Salespeople sold the dog np . (11) ⇒
Salespeople sold the dog biscuits.

La struttura di una frase può essere rappresentata in un albero sintattico

"Salespeople sold the dog biscuits"

np: predicato nominale
vp: predicato verbale
s: sentence

(il seme della grammatica è la radice dell'albero sintattico)



(s-maj s (np (noun *salespeople*)) vp ((verb *sold*) np ((det *the*) *(noun* dog)) (np (noun *biscuits*))) (fpunc .))

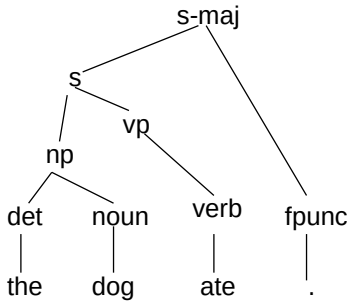
Un'altra frase generata con la stessa CFG

Nota che, con la stessa CFG, si può rappresentare anche la frase:

Regole:

1. s-maj → s fpunc
2. s → np vp
3. np → noun
4. np → det noun
5. vp → verb
6. vp → verb np
7. vp → verb np np
8. det → the
9. noun → salespeople
10. noun → dog
11. noun → biscuits
12. verb → ate
13. verb → sold
14. fpunc → .

"The dog ate"



Estendiamo la Grammatica

Nota che, con la CFG che abbiamo appena definito, non si può rappresentare però la frase:

Regole:

1. s-maj → s fpunc
2. s → np vp
3. np → det noun
4. vp → s verb
5. vp → verb np
6. vp → verb np np
7. np → noun
8. det → the
9. noun → salespeople
10. noun → dog
11. noun → biscuits
12. verb → ate
13. verb → sold
14. fpunc → .

"Jack downloaded a computer game"

(neanche aggiungendo i terminali che occorrono: 'Jack', 'a', 'computer', 'game').

Infatti, 'a computer game' è un np non rappresentabile con la regola 3.

Per rappresentarla, occorre aggiungere alla Grammatica la regola:

np → det noun noun

Nota:

Qual è la differenza fra
"Salespeople sold the dog biscuits"
e
"Jack downloaded the computer game"?

La relazione fra le tre ultime parole :
sold (the dog) biscuits
Vs
downloaded (a computer game

Esercizio 1

Aggiungi, alle regole della CFG del lucido n 4,
la seguente:

np → det noun noun

E prova a disegnare l'albero sintattico della frase:
"Salespeople sold the dog biscuits."

Questo albero è unico?

Le Grammatiche possono essere 'ambigue'

Una *Grammatica* è *ambigua* quando ammette *diverse interpretazioni della stessa frase* (cioè diversi alberi sintattici che la rappresentano).

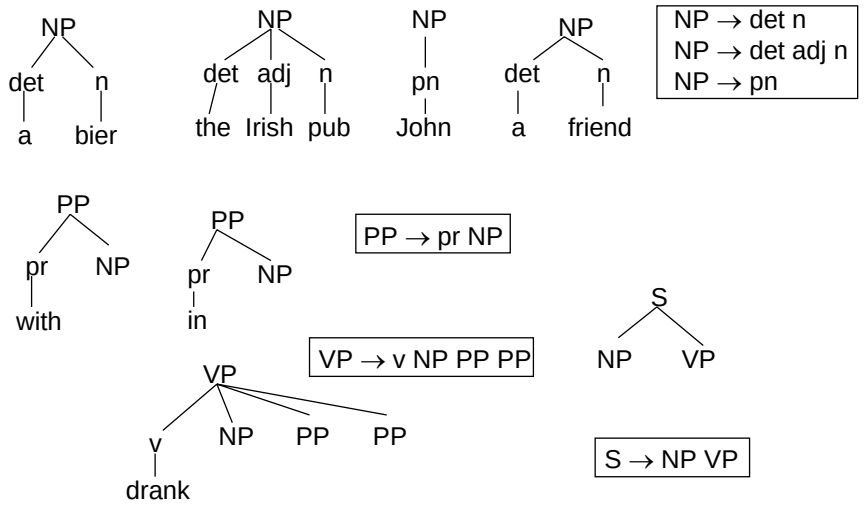
Le Grammatiche devono essere definite in modo da evitare le ambiguità di interpretazione!!! (vedi ad es le Grammatiche dei linguaggi di programmazione).

Tuttavia, alcuni *linguaggi* sono *interentemente ambigui:* Non è possibile, cioè, definire delle grammatiche non ambigue che li rappresentino.

Tutti i *linguaggi naturali* hanno questa caratteristica; vedi l'esempio della frase precedente (comprare 'biscotti per cani' vs 'i biscotti per il cane').

Impariamo a scrivere una Grammatica

Consideriamo la frase: *"John drank a bier in the Irish pub with a friend" .*
Analizziamola a partire dalle 'parts of speech'.



Esercizio 2

Definisci una Grammatica per la generazione delle seguenti frasi relative all'Es 2.2 (dialogo sulla Travel Agency)*:*

I want to go to London
I would like a hotel room
I am leaving from Bari in the morning

Utilità delle Grammatiche

Le Grammatiche sono un modo tipicamente 'generativo' di definire i linguaggi. Possono essere utilizzate, quindi, per definire, in un simulatore di dialogo, il linguaggio in cui si esprime il sistema (generazione 'superficiale' delle singole frasi).

Sono, però, anche alla base dei sistemi di parsing.

Possono essere utilizzate, quindi, anche per definire il linguaggio con cui l'utente può interagire con il sistema (ponendo delle limitazioni al suo modo di esprimersi)

Esempio

Un frammento di grammatica per l'Es 2.2 (Travel Agency):

HOTEL_REQUEST:= WANT HOTEL
WANT:= PRON VERB
PRON:= i / we / …
VERB:= want/ need / would like / …
HOTEL:= hotel / motel / accomodation / place to stay / …

Quali frasi permette di riconoscere questa grammatica?

Esercizio 3

Modifica la grammatica descritta nel lucido precedente per riconoscere anche frasi del tipo:

I would like a nice, quiet hotel in the central part of the city.
I need a cheap room in a hotel for young people.
Please find me a room in a hotel close to Piazza di Spagna!
Hey baby, would you please find me a room in a cheap hotel?

… e simili …

Principi di parsing

Un parser top-down esamina, uno alla volta, gli elementi della stringa di input.

Costruisce e aggiorna una *symbol list (sl)* e un *puntatore di posizione* nella frase.

Pone, inizialmente, il solo seme nella sl, e il valore del puntatore di posizione all'inizio della frase.

Cerca di applicare al primo elemento di sl una delle regole di riscrittura definite nella grammatica.

Quando individua un elemento terminale, sposta in avanti il puntatore e ripete il processo.

Effettua un backtracking quando una sequenza di applicazione delle regole fallisce.

Un esempio semplice di parsing top-down

Grammatica:

1. **s -> np vp**
2. **np -> art n**
3. **np -> art adj n**
4. **vp -> v**
5. **vp -> v np**

"The dogs cried"

The dogs cried
1 2 3

	stato	backup	
1	((s) 1)		
2	((np vp) 1)		regola 1
3	((art n vp) 1)		regole 2 e 3
		((art adj n vp) 1)	
4	((n vp) 2)		matching art with *the*
		((art adj n vp) 1)	
5	((vp) 3)		matching n with *dogs*
		((art adj n vp) 1)	
6	((v) 3)		regola 4
		((v np) 3)	
		((art adj n vp) 1)	
7			Matching v with *cried*

Transition Network (TN)

Le Grammatiche sono un metodo *generativo* per la definizione dei linguaggi.
Permettono cioè di definire *come generare* tutte e sole le stringhe che appartengono ad un linguaggio.

Gli Automi sono, invece, un metodo riconoscitivo per la definizione dei linguaggi.
Permettono cioè di definire come *riconoscere* tutte e sole le stringhe che appartengono ad un linguaggio.

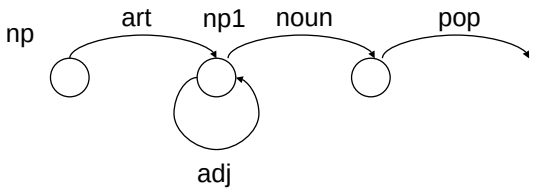
Per il Teorema di equivalenza fra Grammatiche e Automi (nei linguaggi di tipo 3), una Grammatica molto semplice può essere rappresentata, graficamente, con un *Diagramma di Transizione fra gli Stati (TN)*

Transition Network (TN): un esempio

La grammatica G1:
np -> art np1
np1 -> adj np1
np1 ->noun

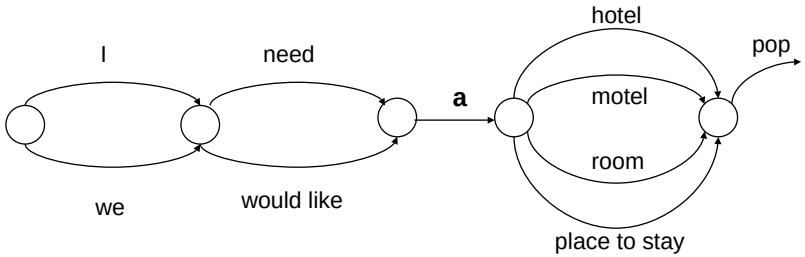
Permette di generare frammenti di frasi molto semplici (predicati nominali) come:
"*a purple cow", "the large American city",…*

Il suo TN equivalente è il seguente:



Agli archi sono associati 'simboli lessicali'

Esempio: il TN di [hotel_request]



Limiti dei Transition Network (TN)

I TN sono automi a stati finiti, equivalenti a 'grammatiche regolari'.

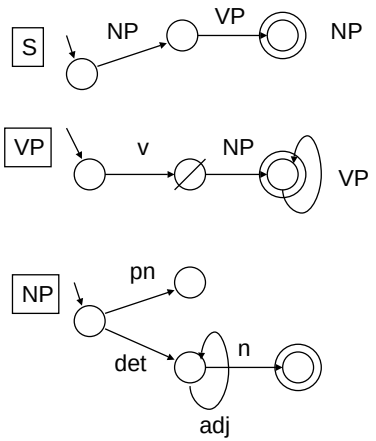
Permettono di fare analisi lessicali o di riconoscere frasi semplici, ma non di riconoscere linguaggi più complessi (come quelli definibili con CFG).

Per questi linguaggi, occorre introdurre una *memoria a stack* e associare ad ogni arco, oltre ad un carattere letto, operazioni di lettura/scrittura in questa memoria.

Occorre cioè passare da 'Transition Network' a 'Recursive Transition Network'

Recursive Transition Networks (RTN)

S'introduce del diagramma di transizione fra gli stati (TN) uno stack che permette di saltare dal TN principale ad uno secondario (innestamento).



John drank
John went to the pub
John went to the Irish pub
John drank a bier in the new Irish pub with a friend
…

Agli archi possono essere associati simboli lessicali oppure non terminali. Quando si attraversa un arco a cui è associato un simbolo non terminale, si memorizza il nome del simbolo in un registro e si salta ad un sub-RTN che ha questo nome. Alla fine dell'esecuzione del sub-RTN, si torna all'RTN principale.

Augmented Transition Networks (ATN)

Una ATN è un diagramma di transizione fra gli stati, ai cui archi possono essere sono associati una label e *una condizione.*

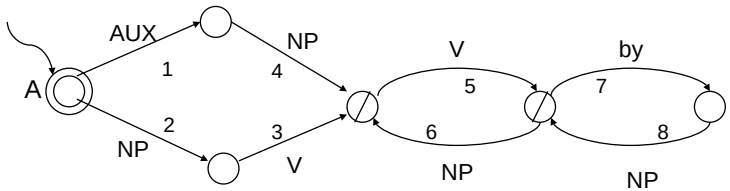
Le *label* possono denotare categorie di parole oppure altre ATN: si tratta quindi di diagrammi ricorsivi.

Le *condizioni* devono essere soddisfatte perché l'arco possa essere attraversato.

Un insieme di *registri* permette di memorizzare risultati intermedi oppure lo stato dell'esplorazione dell'ATN.

Un esempio di ATN

"The rice was eaten by the cat"

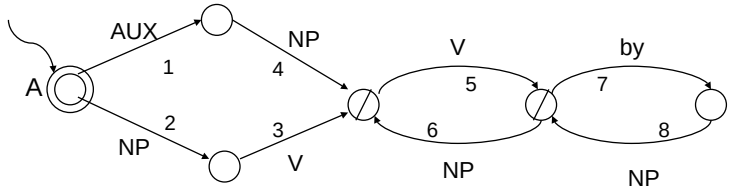


	test	*action*
1	**T**	**SETR V***
		SETR TYPE'QUESTION
2	**T**	**SETR SUBJ***
		SETR TYPE'DECLARATIVE
3	**agrees SUBJ***	**SETR V***
4	**agrees* V**	**SETR SUBJ***
5	**AND(GETF PPRT)(=V'BE)**	**SETR OBJ SUBJ**
		SETR V*
		SETR AGFLAG TRUE
		SETR SUBJ'SOMEONE
6	**TRANS V**	**SETR OBJ***
7	**AGFLAG**	**SETR AGFLAG FALSE**
8	**T**	**SETR SUBJ***

Tabella Condizioni-azioni

Riconosciamo la frase

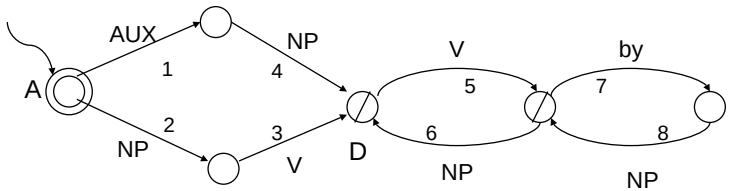
"The rice was eaten by the cat"



Iniziamo dallo start node A.
Ci sono due cammini possibili; ma poiché la prima parte della frase ('the rice') è un NP, si sceglie l'arco 2. Prima di attraversarlo, però, si controlla se ci sono altre condizioni o azioni da soddisfare (vedi tabella). 'T' significa che non ci sono altri test.
Si esegue l'azione che consiste nel memorizzare 'rice' nel registro SUBJECT e TYPE come 'declarative'.
Si passa all'esame di 'was'.
E' un verbo: quindi si traversa l'arco 3. Il test indica di controllare che questo verbo si accordi con il soggetto. Se questa condizione è verificata, si memorizza 'was' nel registro VERB.

Riconosciamo la frase

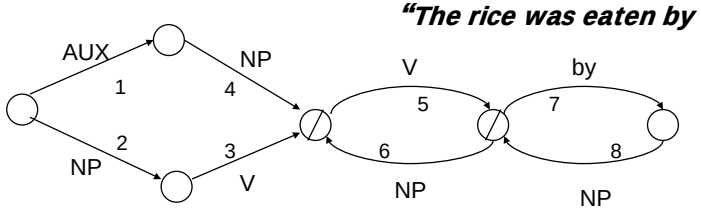
"The rice was eaten by the cat"



Siamo ora al nodo D.
'Eaten' è un verbo; il test sull'arco 5 indica di controllare che questo verbo si accordi con l'ausiliario.
Si compiono ora 4 azioni: il contenuto del registro SUBJ ('rice') viene trasferito nel registro OBJ. Il contenuto del registro VERB viene sostituito con 'was eaten'. Un flag indica che la frase è in forma passiva. Il registro SUBJ viene settato al valore 'someone'.
By: si sceglie l'arco 7.
'The cat': un'altra 'noun phrase': si traversa l'arco 8 e si memorizza il frammento di frase in SUBJECT.
Il processo termina.

Riassumiamo il processo:

"The rice was eaten by the cat"



	test	*action*
•	**T**	**SETR V***
		SETR TYPE'QUESTION
2	**T**	**SETR SUBJ***
		SETR TYPE'DECLARATIVE
3	**agrees SUBJ***	**SETR V***
4	**agrees* V**	**SETR SUBJ***
5	**AND(GETF PPRT)(=V'BE)**	**SETR OBJ SUBJ**
		SETR V*
		SETR AGFLAG TRUE
		SETR SUBJ'SOMEONE
6	**TRANS V**	**SETR OBJ***
7	**AGFLAG**	**SETR AGFLAG FALSE**
8	**T**	**SETR SUBJ***

The rice: **NP; arc 2 (SUBJ = the rice; DECLARATIVE)**

was: **V 3**

eaten: **V 5 (OBJ=rice; V=was eaten; AGFLAG=T; SUBJ=SOMEONE)**

by: **7 AGFLAG=F**

the cat: **NP 8 SUBJ=the cat**

Esercizio 4 (non semplice!)

Interpreta, con l'ATN precedente, le frasi seguenti:

"The cat ate the rice"
"Did the cat eat?"
"The cat saw the TV"
"Was the fish eaten by the cat?"

Fai qualche esempio di frasi non interpretabili con quella ATN.

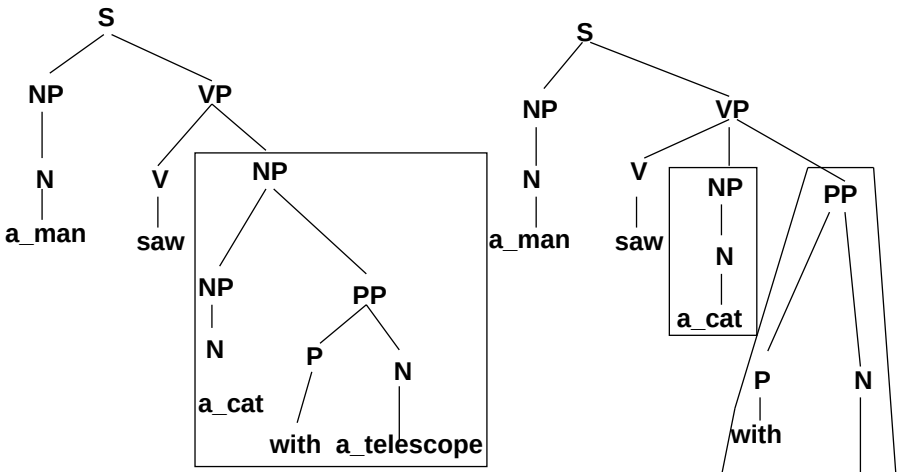
Parsing probabilistico

Non tutte le sequenze di parole si presentano, in un linguaggio, con la stessa frequenza.

Questa conoscenza può essere utilizzata per riconoscere e interpretare frasi in condizioni di incertezza.

In questo caso, il processo di riconoscimento produce, come risultato, un valore di probabilità per ciascuna delle interpretazioni possibili della stringa di input.

Esempio: "A man saw a cat with a telescope"



Qual è l'interpretazione più plausibile?

Quale grammatica

S → NP VP
VP → V NP
VP → V NP PP
NP → NP PP
PP → P N

N → a_man
N → a_cat
N → a_telescope
P → with
V → saw

Grammatiche Probabilistiche

Si propongono di assegnare un valore di probabilità per ogni frase generabile con la grammatica.

Formalmente, la definizione di grammatica probabilistica differisce da quella delle grammatiche tradizionali soltanto nel senso che:

- *ad ogni produzione viene assegnato un valore di probabilità* e
- *la somma dei valori di probabilità associati alle produzioni che hanno, a sinistra, lo stesso non terminale è = 1*.

Esempio:

S → NP VP [.80]
S → AuxNP VP [.15
S → VP [.05]
NP → Det Nome [.20]
NP → NomeProprio [.35]
NP → Nome [.05]
NP → ProNome [.40]

Nome → Noun [.75]
Nome → Noun Nome [.20]
Nome → NomeProprio Nome [.05]
VP → Verbo [.55]
VP → Verbo NP [.40]
VP → Verbo NP NP [.05]
Det → that [.05] the [.80] a [.15]
…

Come usare l'incertezza nelle produzioni

La probabilità di un albero di parsing T per una frase S si può stimare come

$$P(T,S) = \prod_{n \in T} p(r(n))$$

Se diverse interpretazioni sono possibili, l'algoritmo di parsing sceglierà, tra di esse, l'albero con probabilità massima.

Riprendiamo l'esempio precedente

associando ad ogni produzione un valore di probabilità

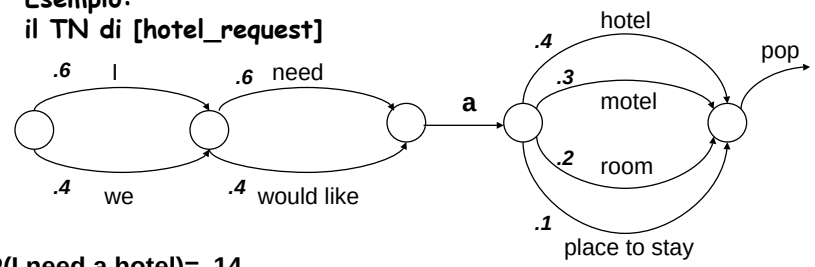
S → NP VP [1]
VP → V NP [.25]
VP → V NP PP [.75]
NP → NP PP [1]
PP → P N [1]

N → a_man
N → a_cat
N → a_telescope
P → with
V → saw

Qual è l'interpretazione più probabile, per questa frase?

E' possibile rappresentare l'incertezza associata a processi di interpretazione di frasi associando valori di probabilità agli archi dei diagrammi di transizione fra gli stati anziché alle produzioni

Esempio:
il TN di [hotel_request]



P(I need a hotel)= .14
P(I need a place to stay)= .04
P(we would like a room) = 03
…

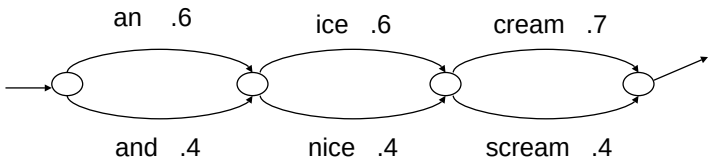
Associamo agli archi delle ATN un valore di probabilità, che indica *"qual è la probabilità di attraversare quell'arco, se ci si trova nello stato da cui l'arco inizia".*

La somma delle probabilità associate agli archi che escono da uno stato deve essere = 1.

Usi delle Probabilistic FSTN

Nella interpretazione dello speech, i Probabilistic FSTN possono essere applicati al riconoscimento di stringhe 'poco chiare'.

Esempio:

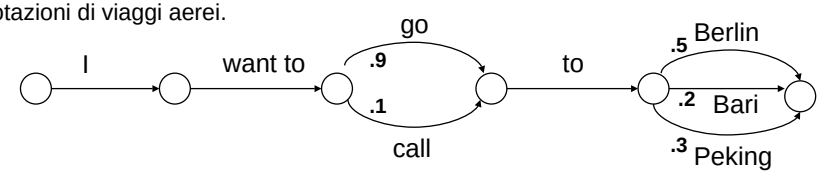


La probabilità di riconoscere una stringa è data dal prodotto delle probabilità associate agli archi attraversati.

P(an ice cream) = .25
P(and nice scream) = .06
P(an nice cream) = .17
…

I Probabilistic FSTN si chiamano anche Catene di Markov

Contesto: sistema automatico di prenotazioni di viaggi aerei.



"I want to go to Berlin": P=.45
"I want to go to Bari": P=.18
"I want to go to Peking": P=.27

"I want to call to Berlin": P=.05
"I want to call to Bari": P=.02
"I want to call to Peking": P=.03

Le Catene Markoviane aiutano a individuare l'interpretazione più probabile di frasi 'incomplete'

Esempio: un estratto di dialogo speech-based In un Call Center.

U: "I want to … to …!"
S: "Do you want to go to Berlin?"
U: "No, to …!"
S: "Oh sorry, it's Peking where you want to go, right?"

Esercizio 5

Fai un altro esempio di Markov Chain per un Call Center per prenotazioni aeree.

Disegna la Catena di Markov e fai un esempio di dialogo.

Semantica

Come abbiamo visto nell'Unità 2, le frasi in linguaggio naturale possono essere formalizzate in un linguaggio logico, allo scopo di 'ragionare' su di esse.

The dog ate.
Ate(dog)

Salespeople sold the dog biscuits
Sold(salespeople, DogBiscuits)
Oppure: Sold(salespeople, Biscuits, Dog),
nelle due interpretazioni della frase che abbiamo visto.

Jack downloaded the computer game
Downoaded(Jack, ComputerGame)

Strumenti

Un esempio di parser del linguaggio naturale può essere scaricato dal sito:

http://cslr.colorado.edu/~whw/phoenix/phoenix_manual.htm

(Phoenix: lavora sotto linux)

Phoenix è parte di Galaxy

Galaxy: "An open source architecture for constructing dialogue systems"

finanziato dal DARPA (la Defense Advanced Research Project Agency americana)

con l'intento di creare interfacce multimediali che integrino speech con grafica, eyetracking, analisi dei gesti ecc.

http://sourceforge.net/index.shtml

I Frame in Phoenix

Phoenix analizza una stringa di input in termini di una sequenza di 'frame semantici'. I frame reppresentano un oggetto o un'azione in un particolare dominio applicativo.

Ad un frame è associata una lista di 'slot name', a ciascuno dei quali è associato un insieme di regole in una grammatica context-free.
Lo slot name corrisponde al *seme* della grammatica (radice dell'albero sintattico).

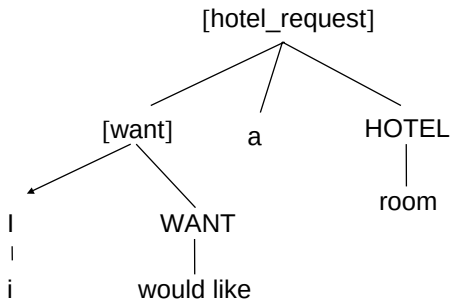
Esempio: Dominio applicativo: informazioni turistiche

FRAME: Hotel
NETS:
[hotel_request]
[hotel_name]
[hotel_period]
[hotel_location]
[room_type]
…

Regole della grammatica

(memorizzate in file .gra)

```
[hotel_request]
        (*[want]* a HOTEL)
HOTEL
        (hotel)
        (motel)
        (room)
        (place to stay)
;
[want]
        (*I WANT)
I
        (i)
        (we)
WANT
        (need)
        (would like)
;
```



Parsing di *I would like a room:*
[hotel_request] ([want](i would like) a room)

Stringhe di minuscole: terminali; stringhe di maiuscole: macro;
stringhe fra **[]**: non terminali

Qualche esempio di file .gra: Anaphora

```
[Anaph]
        (THAT one)
    THAT
        (that)
        (this)
        (THE [_other])
    THE
        (the)
        (any)
        (an)
    ;

    [_other]
        (other)
    ;
```

```
[interj]
        (i guess)
        (i mean)
        (i meant)
        (i'm not sure)
        (i think)
        (*okay LET see)
        (WAIT *a MINUTE)
        (*i say)
        (no matter what)
        (if *IT possible)
        (if *IT available)
        (in any case)
        (wow)
        (hey)
        (then)
        (wait a minute)
        (actually)
```

```
WAIT
        (wait)
        (hold on)
MINUTE
        (minute)
        (second)
LET
        (let's)
        (let *me)
        (let us)
IT
        (that's)
        (it's)
;
```

Social.gra

```
[Polite]
    (please)
    (thank you)
    (thank you very much)
    (thanks *a *lot)
    (you're welcome)
    ;
[Compliment]
    (THATS perfect recognition)
    (THATS marvellous)
    (it seemd to work very well)
    (THATS pretty impressive)
    THATS
        (that's)
        (that was)
    WHAT
        (great)
        (a good job)
        (excellent)
    ;
    [Complaint]
        (hurry up)
        (you dumb machine)
        (you don't do this right)
        (i don't like you)
        (i'm laughing at you)
    ;
    [Greeting]
        (hello)
        (hi)
        (yo)
        (good morning)
    ;
    [Confusion]
        (GENERAL)
        (CHANNEL)
```

```
GENERAL
  (*now i'm confused)
  (we are confused)
  (*i got confused)
  (it's *pretty confused)
  (do you understand what i'm saying)
  (you didn't understand me)
  (can you *still understand me)
  (*i didn't understand that)
  (*i didn't say anything)
  (what are you waiting for)
  (who are you talking to)
  (you're repeating yourself)
  (are you sure about that)
  (*there's something wrong)
  (something must be wrong)
  (there's a problem)
  (this STATE stuck)
  (i'm stuck)
  (you're stuck)
  (it's stuck)
  (are we stuck *again)
  (it's not getting anything)
  (NO_CAN *quite understand *you)
  (NO_CAN understand what you're saying)
  (i don't know what you're talking about)
  (it's not understanding *correctly)
  (it's not doing anything)
  (*now i've got nothing at all)
  (the system has failed)
  (this doesn't work)
  (*it didn't work)
  (completely DEAD)
  (i think it's DEAD)
  (oh shit)
  (damn it)
;
```

```
STATE
    (is)
    (shouldn't be)
DEAD
    (dead)
    (died)
CHANNEL
    (can you hear me)
    (did you hear that)
    (is *there anybody there)
    (*my mic is not on)
    (*are you awake)
    (NO_CAN hear anything)
    (NO_CAN hear you)
    (are you *still there)
    (anybody home)
    (are you listening)
NO_CAN
    (i don't)
    (i didn't)
    (i can't)
    (*i couldn't)
```

Query

```
[i_want]
    (*CAN_ONE [show_me])
    (I'M *ALSO_ONLY *be INTEREST)
    (*FIRST_I'D *ALSO_ONLY
    WANT_NEED_LIKE *to *KNOW)
    (let ME)
    (HOW BOUT)
    (make that)

    ME
        (me)
        (us)
    HOW
        (what)
        (how)

    BOUT
        (about)
        ('bout)

    CAN_ONE
        (CAN ONE)

    INTEREST
        (interested *in)
        (LOOKING *for)

    LOOKING
        (inquiring *about)
        (looking)
        (requesting)

    KNOW
        (know *about)
        (arrange)
        (take)
```

```
WANT_NEED_LIKE
  (have to)
  (like)
  (need)
  (needed)
  (be needing)
  (prefer)
  (prefered)
  (want)
  (wanna)
  (wanted)
  (wish)
  (wished)
  (want to change)
FIRST_I'D
  (*first I'D)
I'D
  (i)
  (i'd)
  (i WILL)
  (i'll)
  (i had)
I'M
  (i'm)
  (i am)
  (i WILL)
  (i'll)
CAN
  (may)
  (can)
  (could)
```

```
WILL
  (will)
  (would)
ALSO_ONLY
  (also)
  (only)
ONE
  (one)
  (we)
  (i)
  (us)
  (you)
;
[is_that]
  (IS THAT)
  (ARE THOSE *[both_all])
  (WILL *[both_all] *of THAT_THOSE *be)
THAT_THOSE
  (THAT)
  (THOSE)
THAT
  (it)
  (that)
  (this)
WILL
  (will)
  (would)
  (can)
  (could)
```

Alcuni possibili usi di Phoenix

Phoenix può essere utilizzato:

nel modo più semplice, definendo frame, slot e regole appropriati per un nuovo dominio applicativo

oppure

estendendo il linguaggio, per domini già definiti nel software (per introdurre, ad esempio, elementi ‘affettivi’ nel dialogo, come nell’ultima frase dell’Esercizio 3).

e infine…

trasformando le grammatiche in grammatiche probabilistiche.